Comparing propagation models of the NS3 network simulator for 802.11n - Lab report for communication on distributed systems

B.Sc. Manuel Faatz

***Abstract*—**

I. Introduction

Network simulation has been a popular tool for network research for a long time. It allows testing new protocols and algorithms in a controlled environment. The NS3 network simulator is a popular open source discrete event network simulator. It is written in C++ and provides a Python interface. It is used in many research projects and is under active development. The NS3 network simulator provides a large number of models for different network layers and protocols. In order to model how signal propagation effects wireless communication, different propagation models have been developed. The NS3 network simulator provides many models for different network layers and protocols. The NS3 network simulator provides many models for different use cases and environments. In this work, some of the ones most applicable for 802.11n have been selected and compared at different distances with respect to throughput and received signal power. Distance is one of the most common factors for Rx power loss in wireless communications, since the spread of the electromagnetic waves across an increasing area with increasing distance means a lower power density at any given point of the area. Additionally, effects like reflection, refraction, absorption and interference can influence the wireless transmission performance. Depending on the propagation model, these effects are simulated in some capacity. However, there is no consensus in the scientific community on which model is most suitable for which type of simulation. Therefore, the author compared a selection of them in a simple setting to have an idea on how each of them stacks up against each other and reality. The scenario chosen for this work is a simple point to point link between two nodes. This ensures that the only factor influencing the signal propagation is the distance between the nodes and the propagation model.

II. Methodology

First of all, suitable propagation models had to be selected. NS3 supports a multitude of propagation models which can be found on [**?**, the ns3 website]. However for this work only the following models were deemed suitable:

- FriisPropagationLossModel
- FixedRssLossModel
- ThreeLogDistancePropagationLossModel
- TwoRayGroundPropagationLossModel
- NakagamiPropagationLossModel

In order to compare them, a ns3 script was set up. It was very similar to the script wifi-spectrum-per-example.cc with some notable changes: Instead of a constant rate Wi-Fi manager (which would make the analysis of the data rate pointless), a Minstrel HT manager was used. Furthermore, callbacks for monitoring the Rx power were added, including some tools for value conversions and averaging. Some command line arguments like simulationTime, disctance, propagationModel, etc. were added as well. Additionally, callbacks were added to the script to save the Rx power and throughput to a file. Both of these saved the time and value of the respective variable in a std::map. At the end of the simulation, both maps were exported to csv files. This would make importing them into pandas trivial. Traffic was generated at the application layer using a UdpClientServerHelper set to a constant data rate of 75 Mbps. Since these simulations required substantial computational resources, they were run on a remote server. For easy experiment control and file management, a python wrapper script was written. It allowed to run an experiment and download all the output files automatically. This wrapper was then imported into a jupyter notebook for testing and data analysis. The data analysis was then performed in the notebook using pandas and matplotlib. The source code for the ns3 script, the python wrapper and the jupyter notebooks can be found on [**?**, the author's github]. If you wish to reproduce the results, please follow the instructions in the README.md file on how to set up ns3.39, symlink the repositories scratch folder into ns3, set up the python environment, set up the environment variables and run the jupyter notebooks. The expectations for the results of the different simulaitons were quite different. The FixedRssLossModel isolates the propagation delay as the only effect of the distance on the transmission. This means that the data rate should be constant until the distance is so large that the propagation delay causes issues with the WiFi protocol timings. This should result in a sudden drop in the datarate as soon as this critical distance is reached. The Rx power per definition should stay constant. For the other models, the data rate should stay constant until the Rx power drops far enough that the current Htmcs value is no longer sufficient to maintain the data rate. This should result in a sudden drop in the data rate down to the datarate assiciated with the next Htmcs value.

III. Results

Since the simulation contains non-deterministic processes and a steady state siimulation is desired, the time until a steady state is reached first had to be determined. Initially, this was attempted by running multimple simulations with gradually increasing simulation time. The result can be seen in figure **??**. However, this approach was

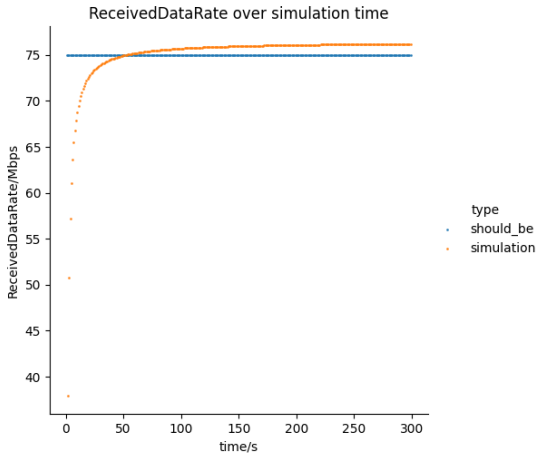

Fig. 1. average datarate over simulation duration

deemed unrealistic. Upon plotting the data rate of a single simulation over time, it was found that the data rate fluctuated heavily in the first 3 seconds and then stabilized. This can be seen in figure **??**.

Keeping this in mind, the first graphic was redone, this time only using the data rate after 3 seconds for each simulation. The result can be seen in figure **??**.

It shows that the average data rate of a simulation is stable after 3 seconds and independent of how much longer the simulation time is running for - which in turn means the the datarate fluctuations seen in figure **??** can be assumed to be uniformly distributed. This

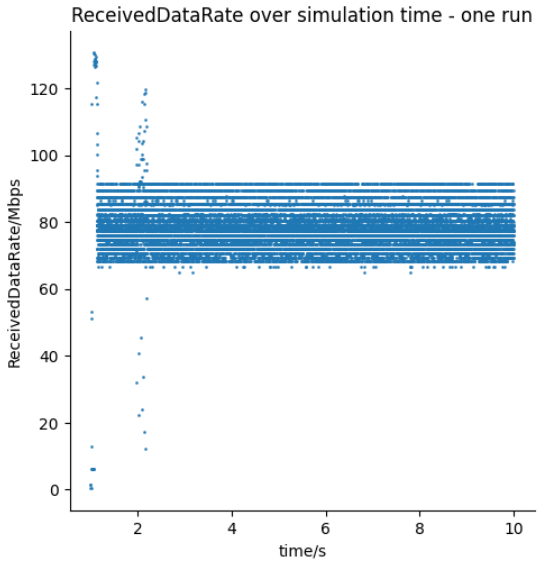

Fig. 2. datarate over simulation time of a single run

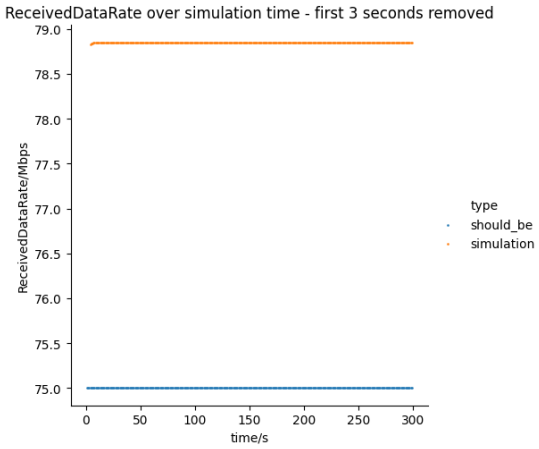

Fig. 3. average datarate over simulation duration without first 3 seconds

means as long as the simulation time is long enough after the 3 seconds to reach a statistically relevant average value, the data rate can be assumed to be independent of the simulation time. With this information, it was decided to run the simulaitons for 10 seconds and only use the output after 3 seconds.

The same procedure was performed for the Rx power. The results of one run can be seen in figure **??**.

For the Rx power, the fluctuations are much smaller. Therefore, it was decided to simply use the average Rx power over the time of the simulation. This meant the averaging could be offloaded to the ns3 script, drastically reducing the computational load of the data analysis.

After determining the steady state, the simulations were run for 10 seconds of simulation time from a distance of 1 meter up to a distance where no more data arrived for each propagation model with a step size of 1 meter.

The results for the data rate can be seen in figure **??**.

The results for the Rx power can be seen in figure **??**.

References

[1] NS-3. (2019) Propagation models. [Online]. Available: https://www.nsnam.org/docs/release/3.39/models/html/propagation.html

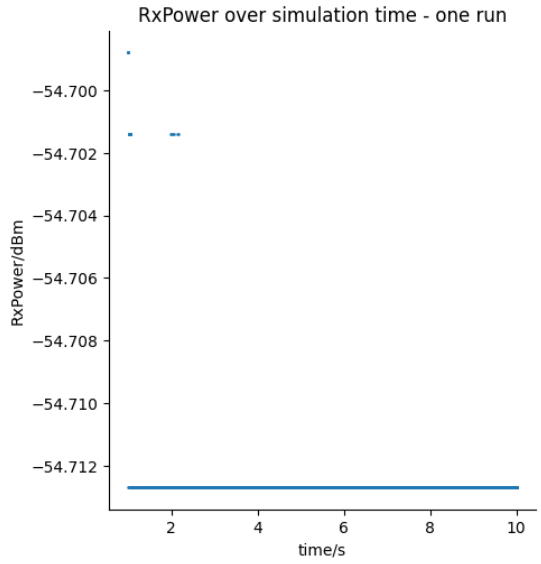

Fig. 4. Rx power over simulation duration

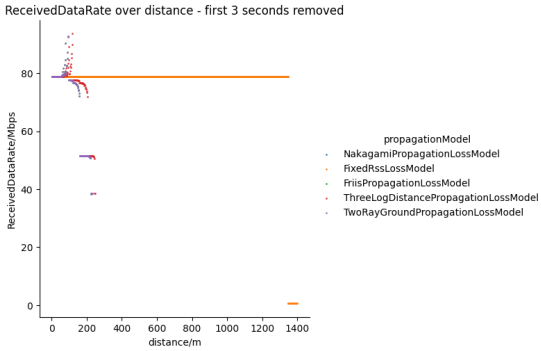

Fig. 5. data rate over distance

[2] M. Faatz. accompanying code. [Online]. Available: https://github.com/MainManu/KVS$_n s3_l ab$

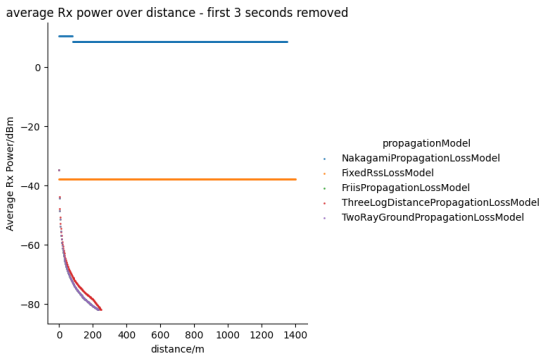

Fig. 6. Rx power over distance